PostScript Primer for Programmers

by Carolyn Scheppner

PostScript is a programming language that provides a hardware-independent standard for representing the printed page. PostScript objects include the Adobe fonts which can be scaled, condensed, expanded, outlined, italicized and rendered anywhere on a page. PostScript can also draw lines and curves to create geometric shapes which may be outlined or filled with any shade. In fact, any arbitrary image can be manipulated with PostScript. Let's take a closer look at these capabilities by trying a few simple examples.

PostScript, like Forth, is a very stack-oriented language. Most PostScript commands take their arguments from the stack and the commands are given using postfix notation - that is, with operators following their operands. Here are a few PostScript statements in postfix notation:

PostScript Statement	Meaning
10 40 moveto	% Move to x,y position 10,40.
	%
(Hi) show	% Print the string "Hi".
	%
/LM 20 def	% Assign the value 20 to variable LM.
	%
3 5 add	% Push 3 then 5 on stack and add them,
	% leaving 8 on stack.

Some operations like "moveto" return no result. Instead they change the state of internal PostScript variables. For instance "moveto" changes your position in the co-ordinate system used by PostScript for drawing operations. This is similar to moving the pen in Logo with the SETXY command. Other operations like "add" and "currentpoint" place results on the stack where subsequent operations can make use of the results. Here is an example which moves to a new position in the PostScript co-ordinate system:

```
currentpoint    % Places the current x,y co-ordinate on the stack
                %   (y on top).
10 add          % Adds 10 to the item on top of the stack
                %   (the y position).
moveto          % Moves to the new x,y co-ordinates which
                %   are on top of the stack (x,y+10).
```

Sometimes the results of an operation are not in the order you need for the next operation. In this case, the operands on the stack can be duplicated, discarded, exchanged or assigned to variables. Here is a longer example which assigns the length of the string "Hello" to a variable:

```
(Hello)         % Pushes the string "Hello" onto the stack.
                %
dup             % Duplicates the top stack entry and pushes
                %     it onto the stack.
stringwidth     % Takes the duplicate entry off of the stack
                %     and uses it to calculate width and height.
                %     Width and height are pushed onto the stack.
pop             % Throw away the height, leaving width on stack.
                %
/strw           % Pushes the variable name strw onto the stack.
                %
exch            % Swaps the top 2 stack items - now the order is
                %       /strw, width.
def             % Assigns the value of width to the variable strw.
                %       (i.e. /strw width def)
```

PostScript control structures are also stack oriented. You create a control structure by pushing a condition on the stack, followed by a procedure, followed by a control operator. The procedure can be any number of statements delimited by the curly braces {}. The example below will print "Is 3", if the variable xyz is equal to 3:

```
xyz 3 eq            % Checks if the variable xyz equals 3. If so, TRUE is
                    %   pushed onto the stack. Otherwise FALSE is pushed.
{(Is 3) show} if    % Executes the statements in curly braces if TRUE is on
                    %   the top of the stack.
```

The use of curly braces is not limited to control statements. You can use them with the "def" operator to group together several operations that define a complex new procedure which you can use just like the built-in operators. This lets you build powerful commands that can be included at the start of your code, allowing your "main" program to be just a sequence of simple procedure calls. Here's an example:

```
% === Variables
/TM  740 def        % set top margin variable
/LM   20 def        % set left margin variable

% === Procedures
/topofpage  { LM TM moveto } def            % move to top left corner
```

This code makes a new "topofpage" command that you can use in your programs. Note that in PostScript co-ordinate system, (0,0) is at the lower left corner of the page, and unless scaled, each unit represents 1/72 of an inch.

Now that you have a better idea of how PostScript works, let's consider a more complicated example. A complete PostScript program which generates a company letterhead is shown below. The example contains a variety of procedures that simplify the selection and styling of fonts and the centering and justification of strings.

If you do not have a PostScript printer, you may want to try the PostScript interpreter by Greg Lee on Fish Disk #101. You can use this to test out your own PostScript code and preview the layout of pages you create. I have included conditional code for Greg's interpreter in the letterhead example. Activate it by setting the InterpFlag variable to 1.

You may have to fiddle a bit with the value I add to the right margin (RM) in the conditional code for the interpreter. I scale the output so that the whole page is compressed to fit on the Amiga's screen, but an adjustment to the right margin variable seems to be needed to match the printer's horizontal layout. Larger font sizes seem to need larger adjustments.

The front end of the letterhead example could be used as a base for your own letterhead, overhead, report cover or sign making programs. Add your own procedures for new features such as graphic elements or shade-filled outline text. To learn more about programming in PostScript, see the PostScript Language Reference Manual or the PostScript Language Tutorial and Cookbook, by Adobe Systems Incorporated.

```
%! PostScript (R) Letterhead Example  by  C. Scheppner
%  PostScript and fontnames are registered trademarks of Adobe Systems Inc.

% === Save the initial state
save

% === Conditional Flag for Greg Lee's ps interpreter, Fish Disk #101
/InterpFlag 0 def        % Set to 1 for interpreter, 0 for printer


% === My margins (RM modified in Interpreter Changes)
/TM 740 def     % top margin
/BM 20 def      % bottom margin
/RM 582 def     % right margin
/LM 20 def      % left margin
```

```
% === Start of Conditional Changes for Interpreter
InterpFlag 1 eq {
2 pencolor
/showpage {
   % do nothing
   } def
/RM RM 260 add def  % fixes spacing somehow
.54 .65 scale       % scale to fit page on screen
erasepage } if
% === End of Changes for Interpreter


% === My Variables initialized to defaults (Note: Margins defined above)
/pagenum 0 def      % Keep track of page numbers
/fsize 20 def       % Font size (height)
/fwidth 20 def      % Font width
/ftilt 0 def        % Font tilt
/outline 0 def      % Outline flag
/cpx LM def         % Currentpoint X for user savecp/restorecp
/cpy TM def         % Currentpoint Y for user savecp/restorecp
/tcpx LM def        % Temp cpx for oshow
/tcpy TM def        % Temp cpy for oshow


% === My Subroutines

% === Save/Restore X/Y position
/savecp    { currentpoint /cpy exch def /cpx exch def } def
/restorecp { cpx cpy moveto } def

% === Perform a newline based on font size
/newline
 { currentpoint fsize sub
   exch pop
   LM exch
   moveto } def

% === Move to top left corner
/topofpage { LM TM moveto } def

% === Move to top left corner, increment pagenum
/newpage
 { topofpage
   /pagenum pagenum 1 add def } def

% === Called by myshow if outline flag is set  (string on stack)
% === Note: stroke kills currentpoint so I must save/adjust/restore
/oshow
  { currentpoint /tcpy exch def /tcpx exch def
    dup stringwidth pop
    tcpx add
    /tcpx exch def
    true charpath stroke
    tcpx tcpy moveto } def
```

```
% ========================  MAIN PROGRAM  ============================
newpage                              % go to top of page
/AvantGarde-Book selectfont          % select a font

36 24 8 setstyle                     % make condensed italic style
(Propeller Soft  ) show              % show this at current position
20 14 0 setstyle                     % smaller condensed non-italic
% show next line right-justified
(Creators of "Editor Wars", the game programmers play) rshow
bigline                              % now call big underline subroutine

14 12 0 setstyle                     % smaller condensed non-italic
newline                              % move down a (size 14) line
savecp                               % save current x/y position for below
newline                              % move down a line
(Max E. Byte) cshowline              % show this centered, then newline
(Programming Genius) cshowline       % show this centered, then newline

12 10 0 setstyle                     % even smaller condensed non-italic
restorecp                            % move back to saved x/y position
(12 Speedbump Lane) showline         % show these 3 lines at left
(West Chester, PA.) showline
(19380   USA) showline

restorecp                            % move back to saved x/y again
(Phone: (900) 555-1212) rshowline % show these 3 lines at right
(FAX: (900) 555-4321) rshowline
(Telex: 5551010110) rshowline

% === Show (print) the page, then restore the initial state
showpage
restore
```

brought to you by
andy finkel